[19] A. Yao, "How to generate and exchange secrets," IEEE FOCS 1986, pp. 162-167.

7 Acknowledgments

We thank Zvi Galil for many helpful discussions, and Benny Chor, Oded Goldreich, and Stuart Haber for useful comments on earlier versions of this paper.

References

[1] J. Bar-Ilan and D. Beaver,
"Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction," ACM PODC 1989, pp. 201-209.

[2] R. Bar-Yehuda, B. Chor, and E. Kushilevitz, "Privacy, additional information, and communication," IEEE Structure in Complexity Theory 1990, pp. 55-65.

[3] D. Beaver, "Multiparty protocols tolerating half faulty processors," Crypto 1989, pp. 560-572.

[4] D. Beaver, "Foundations of secure interactive computing," Crypto 1991.

[5] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, "Security with low communication overhead," Crypto 1990, pp. 62-76.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for noncryptographic fault tolerant distributed computation," ACM STOC 1988, pp. 1-9.

[7] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," ACM STOC 1988, pp. 11-19.

[8] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, "Verifiable Secret Sharing" IEEE FOCS 1985, pp. 383-395.

[9] D. Dolev, C. Dwork, O. Waarts, and M. Yung, "Secret Message Transmissions," IEEE FOCS 1990, pp. 36-45.

[10] T. Feder, E. Kushilevitz, and M. Naor, "Amortized communication complexity," IEEE FOCS 1991, pp. 239-248.

[11] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," ACM STOC 1987, pp. 218-229.

[12] E. Kushilevitz, "Privacy and communication complexity," IEEE FOCS 1989, pp. 416-421.

[13] E. Kushilevitz, "On computing the sum privately," manuscript.

[14] S. Micali and P. Rogaway, "Secure computation," Crypto 1991.

[15] A. Orlitsky and A. El Gamal, "Communication with secrecy constraints," ACM STOC 1984, pp. 217-224.

[16] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," STOC 1989, pp. 73-85.

[17] A. Shamir, "How to Share a Secret," CACM 22 (1979), 612-613.

[18] C. Small, *Arithmetic of Finite Fields*, Monographs and Textbooks in Pure and Applied Mathematics (Vol. 148), Marcel Dekker, Inc., New York, 1991.

Revelation Phase

At the end of the computation phase, each player i has some multi-share v_i, which may be the value at α_i of a degree t polynomial that passes through the real output at 0 and the check output at 1. The players accept the real output if and only if (1) the final multi-shares lie on a polynomial of degree at most t; (2) all values shared using VSS were shared honestly; and (3) the check output matches the result of the computation on the check inputs.

Undetected cheating occurs if at least one multiplication subprotocol is tampered with, and the final check output is unaffected. As stated earlier, this occurs with probability at most $\frac{Mn(n+1)}{|F|^{m-1}}$, which is less than ϵ for $m = \lceil 1 + \frac{\log(Mn(n+1)\epsilon^{-1})}{\log|F|} \rceil$. With this value of m, the size of each message in the protocol is $m \log|F| = O(\log(|F|Mn\epsilon^{-1}))$, from which the communication complexity follows. □

We note that detectability implies resilience, assuming that faulty behavior is infrequent. Define a p-adversary to be an adversary that interferes with the protocol with probability p. The detecting protocol described above can be converted into a *resilient* protocol by simply repeating it until no faults are detected. For a p-adversary, this increases the bit complexity by an expected factor of p^{-1}:

Corollary 1 *For $n \geq 3t+1$, $c \in C_M$, there is a $(t-1)$-resilient protocol to compute c versus a p-adversary with error probability $\frac{p\epsilon}{1-p+\epsilon}$ at an expected cost of $O((n^5 + Mn^2p^{-1})\log(|F|Mn\epsilon^{-1}))$ bits of communication.*

We also note that the basic protocol can be modified to accommodate $z > 1$ check inputs. The probability of successful cheating reduces exponentially with z, the cost of initialization increases multiplicatively with z, and the number of faulty processors that can be tolerated decreases additively with z.

6 Conclusion

We have initiated the study of the communication complexity of multi-party non-cryptographic secure computation protocols. Parallelization can often reduce the amortized bit complexity of existing protocols at a small price in fault tolerance. Allowing no reduction in fault tolerance, matching lower and upper bounds show that amortized bit complexity can be improved in one setting (privacy) but not in another (unstoppability). When malicious behavior is infrequent, detectability can offer as much protection as resilience, at significantly reduced bit complexity.

Many open questions remain in the general area of communication complexity for secure computation. Non-trivial lower bounds for resilient computation and for cryptographic protocols (e.g., allowing one-way functions or Oblivious Transfer) would be interesting. Tradeoffs among complexity measures also warrant further study. We have shown a tradeoff between bit complexity (multiplicative) and fault tolerance (additive); it is possible that other measures can be related as well (e.g., round complexity, level of security, error probability). Lastly, the resilient protocol of Beaver, Feigenbaum, Kilian, and Rogaway [5] has a bit complexity that is *independent* of circuit size, but is low in fault tolerance (i.e, can only tolerate $O(\log n)$ faulty players); whether fault tolerance can be improved while maintaining this independence remains open.

Theorem 6 *For $n \geq 3t+1$, $c \in C_M$, there is a $(t-1)$-detecting protocol to compute c with error probability ϵ, with $O((n^5 + Mn^2)\log(|F|Mn\epsilon^{-1}))$ bits of communication.*

Proof : The protocol actually performs the computation of c over an extension field F' of F, where $|F'| = |F|^m$ for some m. The actual choice of m that will make the error probability less than ϵ will be given at the end of this proof sketch.

The protocol has an initialization phase in which each secret input is multi-shared along with one "check" input. Each check input is random, and jointly created by all players. The verifiable secret sharing [8] (VSS) due to Ben-Or, Goldwasser, and Wigderson [6] is used in this stage, at a communication cost of $O(n^3 \log|F'|)$ bits per VSS.

After initialization, the *private* protocol from Theorem 2 is performed, with some modifications. At the end of the computation, there is a revelation phase, in which the real secret output, the check output, and all check inputs are revealed to all players. If the check computation is wrong, then the honest players reject the real secret output.

Initialization Phase:

1. Each player i finds an otherwise random degree $t-1$ polynomial $p_i(x)$ such that $p_i(0) = s_i$, and gives to each player j the share $s_{ij} = p_i(\alpha_j)$ using VSS.

2. Each player i finds n random degree $t-1$ polynomials $r_{i1}(x), \cdots, r_{in}(x)$, and gives to each player j the shares $r_{i1j} = r_{i1}(\alpha_j), \cdots, r_{inj} = r_{in}(\alpha_j)$ using VSS. The lth "check input" will be $\sum_{i=1}^{n} r_{il}(1)$.

3. Each player i finds $v_{1i}, \cdots, v_{ni}$ where $v_{li} = (1-\alpha_i)s_{li} + \alpha_i(\sum_{j=1}^{n} r_{jli})$. These values are player i's multi-shares of n polynomials of degree t that pass through the secret inputs at 0 and the check inputs at 1.

Computation Phase

The computation phase proceeds exactly as described in the protocol for Theorem 2 (on *two* sets of inputs), except that in the truncation step of the multiplication subprotocol each player i now gets the value v_i where $(v_1 \cdots v_n) =$

$$\sum_{j=1}^{2} \vec{w} B_{e_3}^{-1} Chop_{2t} B_{e_3} B_{e_j}^{-1} Chop_{t-1} B_{e_j} M_{e_j}.$$

Here $e_1 = 0$ (secret location of real input), $e_2 = 1$ (secret location of check input), and e_3 is a random field element created jointly by the players during truncation (immediately after the w_i values have been shared) via a basic private addition protocol for random inputs.

This modification insures that cheating in the multiplication will either be detected (e.g., if an attempt is made to influence the choice of e_3), or will perturb the value of the polynomial at e_2 with large probability (i.e., if the adversary fails to anticipate the value of e_3). If cheating occurs in a single multiplication subprotocol that is not immediately detected, then the output of the circuit for random inputs is unchanged with probability at most $\frac{n(n+1)}{|F|^{m-1}}$ (derived from a bound on the number of zeros of the $(n+1)$-variable multinomial that computes the difference between the garbled circuit and the real circuit). Cheating in all M multiplications cannot increase this probability to more than $\frac{Mn(n+1)}{|F|^{m-1}}$.

Lemma 5 *(Parallel Unstoppable Lower Bound) Perfect oblivious t-unstoppable addition among $n = 2t + 1$ players k times in parallel requires $\Omega(kn^2 \log|F|)$ bits of communication.*

Proof : As with the upper bound proof (by the computation over the extended field of size m^k and the additivity of the log function), it suffices to show an $\Omega(n^2 \log|F|)$ lower bound for a single addition. Assume that fewer than $n(n-1)\log|F|$ bits are sent. Then some pair of players p and p^* send fewer than $2\log|F|$ between themselves. Partition the rest of the players (excluding p and p^*), into a subset D of t players and a subset C of the remaining $t-1$ players.

Consider the following execution of the protocol. The players in C drop out immediately, sending no messages, and player p^* drops out after the commit point. For such an execution, the messages sent between p and p^* are a function only of the secret inputs s, s^*, s_D and random tapes r, r^*, r_D of p, p^*, and D.

Let $comm_{p,p^*}$ denote the set of possible communications between p and p^*. If we fix the secret inputs and random tapes of D, then each communication pattern in $comm_{p,p^*}$ is consistent with at most one setting of s and s^*. Otherwise, there would exist two settings s_1, s_1^*, r_1, r_1^* and s_2, s_2^*, r_2, r_2^* for which the communication between p and p^* would be identical. But then the communications between p and p^* would also be identical for the two settings s_1, s_1^*, r_1, r_1^* and s_1, s_2^*, r_1, r_2^*, since each player's view of the ongoing protocol would be unchanged. This violates the assumption of perfect t-unstoppability, since it implies that D and p together cannot determine p^*'s secret input, and thus cannot determine the sum of all players remaining after the commit point, with zero probability of error.

Since $|comm_{p,p^*}| < |F|^2$, there must be some setting of s and s^* that is not consistent with any communication between p and p^*, for fixed s_D, r_D. Thus D can learn some information about the secret inputs of p and p^*, i.e., that at least one possible setting of those secrets is impossible. This violates the assumption of perfect t-unstoppability. □

These two lemmas are summarized by the following theorem.

Theorem 5 *(Negative Direct Sum Result) Parallelization cannot reduce the communication complexity for perfect oblivious t-unstoppable addition (over a finite field) among $n = 2t + 1$ players.*

5 Communication complexity of detecting protocols

Protecting fully against malicious behavior can be quite expensive. For example, the lowest bit complexity for t-resilient computation, where t is a constant fraction of n, is due to Ben-Or, Goldwasser, and Wigderson [6]: $O(Mn^5 \log|F|)$ bits to t-resiliently compute a circuit in C_M (when $n \geq 3t + 1$). However, if malicious behavior is not frequent, then it may be sufficient to merely identify cheating when it occurs. Repeating the computation when cheating is detected, possibly switching to a resilient protocol, insures that all computations are eventually done correctly. This is advantageous if computation that identifies cheating can be more efficient than computation that corrects cheating.

In fact, t-detectability, where t is a constant fraction of n, and M is non-constant, can be achieved at a significant decrease in communication complexity. The protocol that satisfies the following theorem has a bit complexity that is within a log factor of the most efficient known protocol for general t-private computation (when t is a constant fraction of n), when M is sufficiently large.

a group D of at most t players can learn all inputs and all outputs to the probabilistic TM that controls player u.

Assume that a random tape of length k is sufficient for each player in the protocol. The group D can check all possible 2^{k+1} settings of u's input bit and random tape. For each setting, if the TM for player u outputs the exact sequence of s sent messages when it receives the exact sequence of r received messages, then that setting is said to be consistent.

If every consistent setting involves the same choice of input bit for u, then D can learn that secret input. Thus the adversary can learn the input bit of u with probability at least $\frac{(n-t)!t!}{n!}$, i.e., by correctly guessing which t players to corrupt (or with probability 1 if the protocol is oblivious). This violates the zero probability of disclosure needed for perfect privacy. If different consistent settings involve different input bits for u, then the protocol cannot guarantee the zero probability of output error needed for perfect privacy. □

Lemma 3 *(Parallel Private Upper Bound) Perfect t-private addition (mod 2) among $n \geq 2t+1$ players $n-t$ times in parallel is possible with $O(n^2 \log n)$ bits of communication.*

Proof : Each player multi-shares its $n-t$ secret input bits using the $(t, n; n-t, n)$-multi-secret sharing scheme over $GF(2^m)$, where $2^m > 2n - t$ ("room" for n share locations and $n-t$ secret locations). Each player takes the xor of all received shares, and these results are interpolated to recover all $n-t$ xors. The proof follows from Theorem 1 and Lemma 1. □

We summarize these two lemmas in the following theorem.

Theorem 4 *(Positive Direct Sum Result) Parallelization can reduce the amortized communication complexity of perfect t-private addition over $GF(2)$ among $n \geq 2t+1$ players by a factor of $\Theta(\frac{n}{\log n})$.*

We note that the technique of Lemma 3 applies to any finite field, implying that t-private addition over $GF(q)$ can be performed $k \leq n-t$ times in parallel among $n \geq 2t+1$ players with $O(n^2 \log(nq))$ bits of communication. We also note that essentially the same result as Lemma 2 was found independently by Kushilevitz [13], who gives exactly matching upper and lower bounds of $\frac{n(t+1)}{2}$ messages for multi-party private addition.

However, performing in parallel many unstoppable addition protocols over any finite field, without reducing other parameters, cannot decrease the required amount of communication, as the following two lemmas indicate.

Lemma 4 *(Parallel Unstoppable Upper Bound) Perfect oblivious t-unstoppable addition among $n = 2t+1$ players k times in parallel is possible with $O(kn^2 \log |F|)$ bits of communication.*

Proof : The protocol BGW-priv, when used to perform a single addition, is perfect and oblivious. It is actually t-unstoppable, and requires $O(n^2 \log |F|)$ bits of communication. The proof of the lemma then follows from the equivalence of performing k additions over a field of size m and performing a single addition over a field of size m^k (since $GF(m^k)$ can be constructed as $GF(m)[X]/f(X)$), where $f \in GF(m)[X]$ is any irreducible polynomial of degree k [18]). As with the proof of Lemma 3, $GF(m^k)$ must be large enough to have n share locations and one secret location, i.e., $|F|$ must be $\Omega(n^{1/k})$. □

multiplication, k circuits can be $(t-k+1)$-resiliently computed at no increase in communication complexity, by substituting our multi-sharing scheme for Shamir secret sharing.

Other examples where systematic parallelization applies are the private and resilient protocols of Chaum, Crépeau, and Damgård [7], the constant-round protocols of Bar-Ilan and Beaver [1], and the general network topology protocols of Dolev, Dwork, Waarts, and Yung [9].

3.5 Computing "Similar Circuits" in Parallel

We also note that our parallel techniques are useful when circuits that are similar but not identical are to be computed on different sets of inputs. We have inexpensive protocols to split and join multi-shares. At roughly the cost of a single 2-ary multiplication subprotocol, multi-shared secrets can be converted into singly shared secrets, and singly shared secrets can be converted into multi-shared secrets, via secure multi-party protocols (either private or resilient). Thus the computations can be performed in parallel over those portions of the circuits that are identical, and performed separately over those portions that differ. If the circuits can be matched except for portions containing a small number of multiplication gates, then parallelization can still reduce communication complexity.

For example, suppose that the players want to join k singly-shared secrets $g_1(e'_1), \cdots, g_k(e'_k)$, where each player i begins with the shares $g_1(\alpha_i), \cdots, g_k(\alpha_i)$. The goal is to give each player i the multi-share $v_i = g(\alpha_i)$, where $g(e_j) = g(e'_j)$ for all j, $1 \leq j \leq k$. The algebra underlying the joining protocol is that

$$\vec{v} = \sum_{j=1}^{k} \vec{w} B^{-1}_{e'_j} \times Chop_{t-k+1} \times B_{e_j} \times M_{e_j}$$

where

$$w_i = \sum_{j=1}^{k} L_i(e'_j) g_j(\alpha_i).$$

4 Direct Sum Problem for Communication Complexity of Secure Protocols

In the preceding section, we described a general technique for reducing the amortized bit complexity of a secure computation protocol, but at a small cost in number of faults tolerated. It is natural to consider whether this cost is necessary, i.e., can a reduction in amortized bit complexity ever be achieved without affecting any other parameters? This is a version of the "direct-sum problem." The direct-sum problem for communication complexity has been considered recently by Feder, Kushilevitz, and Naor [10]. We present the first such results, one positive and one negative, for this problem for *secure computation*. We concentrate on the addition function.

Performing in parallel many private addition protocols over $GF(2)$ can decrease the required amount of communication at no cost to other parameters, as the following two lemmas indicate.

Lemma 2 *(Non-Parallel Private Lower Bound) Perfect t-private addition (mod 2) among $n \geq 2t+1$ players, where t is $O(n)$, requires $\Omega(n^2)$ bits of communication.*

Proof : It suffices to show that at least $nt/2$ messages must be exchanged. If fewer messages are exchanged, then some player u receives r messages and sends s messages, where $r + s \leq t$. Thus

otherwise. This linear computation can also be performed using the known subprotocol for *sequential* linear operations: Singly share the pseudo-multi-shares, and perform the appropriate linear computations.

By making the systematic modifications described above, we derive a protocol whose properties are summarized by the following theorem.

Theorem 2 *(Private Computation on Multi-Shares) For $n \geq 2t+1$, $t \geq k \geq 1$, there is a $(t-k+1)$-private protocol to compute any arithmetic circuit $c \in C_M$ on k sets of inputs in parallel at a total cost of $O((M+1)n^2 \log |F|)$ bits of communication.*

Proof : Correctness of the protocol sketched above follows from the algebra. The communication bound follows since the initial multi-secret sharing requires $O(n^2 \log |F|)$ bits, and each multiplication subprotocol (both re-randomization and truncation) requires $O(n^2 \log |F|)$ bits as well. Privacy follows from Theorem 1, together with the claim that the multiplication subprotocol reveals no useful information to any coalition of t players. In that subprotocol, each player receives a point on a polynomial of the form $\sum_{j=1}^{k} L_{e_j}(x)h(e_j) + q(x)\prod_{j=1}^{k}(x-e_j)$, where $q(x)$ is a random polynomial of degree $t-k$, and where $h(e_i)$ is the product of secret values for $1 \leq i \leq k$. From $t-k+2$ of these points, the value of a linear combination of $h(e_1), \cdots, h(e_k)$ could be determined. Any smaller set of points can only yield a linear combination of unknowns that includes one or more of the random coefficients of $q(x)$. □

3.4 Other Parallel Protocols

Many other secure computation protocols that rely on Shamir's secret sharing scheme can be parallelized by modifying the algebra to accommodate multi-shares. In fact, all known non-cryptographic protocols tolerating a constant fraction of faulty players are parallelizable by our technique. In all cases, a multiplicative factor of k, for any $1 \leq k \leq t$, is gained in the amortized bit complexity, at the expense of an additive factor of k in the number of faulty players tolerated.

In particular, Ben-Or, Goldwasser, and Wigderson [6] present a t-resilient perfectly-secure protocol (BGW-res) for computing any arithmetic circuit in C_M over finite field F by $n \geq 3t+1$ players with communication complexity $C_{BGW}(C_M) = O(Mn^2t^3 \log |F|)$ bits.

Theorem 3 *(Resilient Computation on Multi-Shares) For $n \geq 3t+1, t \geq k \geq 1$, there is a $(t-k+1)$-resilient protocol to compute any arithmetic circuit $c \in C_M$ (over a finite field) on k sets of inputs in parallel, at a communication complexity $C_{BGW}(C_M)$.*

The BGW-res protocol adds verification and error-correction procedures to the protocol described in the preceding subsection. The verification procedures are always performed on single shares of single shares, which do not require modification when parallelized; these remain the same when performed on single shares of multi-shares. The error correction procedure is also independent of the number of secrets hidden by the polynomial; as long as the α_i's are chosen to be nth roots of unity, this procedure need not be changed for the parallel version.

Rabin and Ben-Or [16] (see also Beaver [3]) have a protocol to securely compute any arithmetic circuit t-resiliently, where $n \geq 2t+1$, assuming broadcast, and allowing an exponentially small probability of error. With systematic changes to the truncation and randomization steps for

is thus also a local operation by each individual player. These nice homomorphic properties of multi-shared secrets are summarized in the following lemma.

Lemma 1 *(Homomorphic Multi-Shares) Any linear combination of multi-shares is a multi-share of the linear combinations of multi-shared secrets, providing that the same secret locations and the same share locations were used for all multi-shares.*

3.3 Parallel Private Computation

In this section, we illustrate our parallelization technique by showing how the t-private arithmetic circuit computation protocol due to Ben-Or, Goldwasser, and Wigderson [6] (which will be called the "BGW-priv" protocol), can be parallelized. Multi-secret sharing replaces Shamir's single-secret sharing scheme, and the algebra is extended to manipulate multi-shares appropriately. A multiplicative factor of k in the amortized communication complexity is gained; an additive factor of k in the number of faulty players tolerated is required. The same ideas can be applied to the private computation protocol due to Chaum, Crépeau, and Damgård [7] (over fields of the proper form).

In BGW-priv, each player shares its secret input using Shamir's scheme. In our parallelization, each player multi-shares its k inputs using the $(t-k+1, t+1; k, n)$-multi-sharing scheme from the proof of Theorem 1. In BGW-priv, linear computations are simple, due to homomorphic properties of shared secrets. By Lemma 1, the same homomorphic properties of multi-shares can be exploited for linear computations.

It suffices to show how the BGW-priv subprotocol for 2-ary multiplication can be parallelized. In BGW-priv, if each player multiplies its shares of two secrets, the result is a "pseudo-share" of the product of the secrets, i.e., it is necessary to re-randomize the polynomial and to reduce its degree from $2t$ to t. In our parallelization, if each player multiplies its two multi-shares, the result is a "pseudo-multi-share" of the k products, similarly failing to be a true multi-share because the underlying polynomial is non-random and degree $2t$.

In BGW-priv, re-randomization is achieved by having each player distribute values of an otherwise random degree $2t$ polynomial which is zero at the single secret location, whereupon each player adds all received values to its pseudo-share. This randomizes the underlying polynomial without affecting its value at the single secret location. For our parallelization, it suffices that the polynomial each player distributes be zero at *all* secret locations.

In BGW-priv, the degree reduction is achieved by noting that it is a linear operation. Specifically, if w_i is the non-degree-reduced pseudo-share of player i, and v_i is the degree-reduced share of player i, then $(v_1 \cdots v_n) = (w_1 \cdots w_n)A$, for some publicly know constant matrix A. More specifically, $A = B^{-1} Chop_t B$, where B is the n by n vandermonde matrix whose (i,j) entry is α_j^{i-1}, and where $Chop_t$ is the n by n matrix whose (i,j) entry is 1 if $1 \leq i = j \leq t$ and 0 otherwise. Thus degree reduction can be achieved by using the known subprotocol for any linear operation: Singly share the pseudo-shares, and perform the appropriate linear computations.

For our parallelization, degree reduction is also a linear operation. If w_i is the non-degree-reduced pseudo-multi-share of player i, and v_i is the degree-reduced multi-share of player i, then again $(v_1 \cdots v_n) = (w_1 \cdots w_n)A$ for a constant matrix A. In this case, $A = \sum_{l=1}^{k} B_{e_l}^{-1} Chop_{t-k+1} B_{e_l} M_{e_l}$, where B_{e_l} is the n by n vandermonde matrix at e_l whose (i,j) entry is $(\alpha_j - e_l)^{i-1}$, $Chop$ is as before, and M_{e_l} is the n by n matrix whose (i,j) entry is $L_i(e_l) = \frac{\prod_{i' \neq l} (\alpha_i - e_{i'})}{\prod_{i' \neq l} (e_l - e_{i'})}$ if $i = j$ and 0

$k < t$ sets of inputs tolerating $t - k + 1$ faulty players. The round complexity, bit complexity, and level of security are unchanged.

Recall that we consider fault-tolerant protocols that can withstand a constant fraction of faulty players (typically $n \geq 2t + 1$ or $n \geq 3t + 1$). For these protocols, the parallel version can compute f at $\Theta(n)$ sets of inputs while still tolerating a constant fraction of faulty players.

3.1 Multi-Secret Sharing

A "multi-secret sharing scheme" lies at the heart of our general compilation technique for parallelizing secure protocols. In this subsection, we define multi-secret sharing and present our scheme.

A $(c, d; k, n)$-multi-secret sharing scheme is a protocol between a Dealer and n players with a Distribution Phase and a Recovery Phase, such that (1) the Dealer begins with k secrets $s_1, \cdots, s_k$; (2) the Dealer sends a message to each player in the Distribution Phase; (3) any subset of at least d players can reconstruct all k secrets in the Recovery Phase from their received messages; and (4) no subset of at most c players can deduce anything, in an information-theoretic sense, about the k secrets from their received messages. The message sent from the Dealer to a player in the Distribution Phase is called a "multi-share" of the k secrets.

Theorem 1 *There is a $(t - k + 1, t + 1; k, n)$-multi-secret sharing scheme to share k elements of a finite field F among n players, $k \leq t < n$, where each multi-share is a single element of F.*

Proof : Let $s_1, \cdots, s_k$ be the Dealer's secrets. Assume that $\alpha_1, \cdots, \alpha_k$ and $e_1, \cdots, e_k$ are preselected elements of F that are known to the Dealer and all n players.

A generalization of Shamir's secret-sharing scheme [17] suffices. In the Distribution Phase, each player i receives the multi-share $p(\alpha_i)$, where $p(x) \in F[x]$ is an otherwise random degree t polynomial such that $p(e_i) = s_i$, $1 \leq i \leq k$. More specifically, $p(x) = q(x) \prod_{i=1}^{k}(x - e_i) + \sum_{i=1}^{k} s_i L_i(x)$, where $q(x)$ is a completely random degree $t - k$ polynomial, and where $L_i(x)$ is the Lagrange polynomial $\frac{\prod_{j \neq i}(x - e_j)}{\prod_{j \neq i}(e_i - e_j)}$.

Any $t + 1$ players can interpolate their multi-shares to recover $p(x)$, and hence recover all k secrets. For any $t - k + 1$ multi-shares, there is a single polynomial that is consistent with those multi-shares and *any* k secrets. □

When secrets are shared using this protocol, and when the parameters c, d, k, and n are clear from context, we may say that the k secrets have been "multi-shared" at "secret locations" $e_1, \cdots, e_k$, using "share locations" $\alpha_1, \cdots, \alpha_k$, and that $p(\alpha_i)$ is the "multi-share" of the secrets received by player i.

3.2 Homomorphic Multi-Shares

If $m_1, \cdots, m_n$ are multi-shares of secrets $s_1, \cdots, s_k$, then $cm_1, \cdots, cm_n$ are multi-shares of $cs_1, \cdots, cs_k$; scalar multiplication of multi-shared secrets can thus be performed by the individual players by a private operation on each individual multi-share. If $m'_1, \cdots, m'_n$ are multi-shares of secrets $s'_1, \cdots, s'_k$, and if the share locations and secret locations are the same for both sets of multi-shares, then $m_1 + m'_1, \cdots, m_k + m'_k$ are multi-shares of $s_1 + s'_1, \cdots, s_k + s'_k$; addition of multi-shared secrets

player begins with a secret input on its private tape. At the end of a computation protocol, each player has on its output tape the value of the function at the secret input values. Unless otherwise stated, all protocols in this paper are assumed to be synchronous (rounds of communication interleaved with periods of private computation), have *oblivious* message-size (number of bits sent in round i from player j to player k depends only on i, j, and k), and perfect (have zero probability of being insecure or computing with error – sometimes we realx this to be "almost perfect").

A protocol is *t-private* if no collection of t players, while following the protocol exactly, gains any additional information about the other $n-t$ secret inputs beyond what is necessarily revealed by knowing their t secret inputs and the output[2].

A protocol is *t-resilient* if no collection of t players, while possibly violating the protocol in an arbitrary and coordinated manner, either gains any additional information about the other $n-t$ secret inputs or prevents the honest players from learning an appropriate output of the protocol. Here an appropriate output is somewhat difficult to define (see, e.g., Beaver [4] and Micali and Rogaway [14] for more precise definition).

We introduce the new and natural security requirement of *t-unstoppability*. A protocol is t-unstoppable if there is a commit point in the protocol such that no collection of t players, while possibly dropping out of the protocol, either gains any additional information about the other $n-t$ secret inputs or prevents the honest players from learning an appropriate output of the protocol. Here an appropriate output would be the value of the function with default values substituted for all players that dropped out before the commit point. This definition captures the fail-stop behavior, which is actually provided by various protocols in the literature.

We also introduce the new and natural security property of *t-detectability*. A protocol is t-detecting if no collection of t players, while possibly violating the protocol in an arbitrary and coordinated way, can either learn any additional information about the other $n-t$ secret inputs, or prevent the honest players from detecting (with high probability) that tampering has occurred. While resilience provides us with correcting capabilities, detectability implies only that violations are observed.

3 Compilation Technique for Parallelizing Secure Protocols

In this section, we present a systematic technique to parallelize many known secure computation protocols. For any k, $1 \leq k \leq t$, this technique allows a single function to be computed simultaneously at k distinct sets of inputs. The communication complexity remains the same as in the original sequential version, for an amortized savings of a multiplicative factor of k. The number of rounds remains the same, and the information-theoretic level of security is maintained throughout. The *multiplicative* gain in bit complexity comes at the expense of an *additive* factor of k in the number of faulty players that can be tolerated, i.e., from t for the original sequential version to $t-k+1$ for the parallelized version.

Thus, for many secure computation protocols in the non-cryptographic setting, the following "meta-theorem" applies:

Meta-Theorem 1 *(Generic Parallel Compilation) A protocol that computes f at a single set of inputs tolerating t faulty players can be systematically modified into a protocol that computes f at*

[2]In this and subsequent definitions, the coordinated behavior of faulty players can be assumed to be under the control of a single (probabilistic Turing Machine) adversary.

and develop techniques to achieve it at a significant decrease in bit complexity over resilience.

1.1 Organization of the Paper

In Section 2, we present the basic model for secure computation and the basic fault models. In particular, we offer two security notions new in the setting of secure computation: *unstoppability* (to withstand fail-stop mode of failure) and *detectability* (to detect arbitrary deviation from the specified protocol). These requirements are both in between the known "privacy" and "resilience" modes.

In Section 3, we describe our general technique for parallelizing non-cryptographic computation protocols, at a small cost in fault-tolerance. Our technique replaces polynomial-based (single) secret sharing with a technique allowing multiple secrets to be hidden in a single polynomial. We illustrate the "multi-shares" technique on a protocol for general computation versus a passive adversary. The technique actually applies to all of the protocols for secure computations which use polynomial-based threshold schemes and applies to all fault-tolerance models.

In Section 4, we prove matching lower and upper bounds to answer two direct sum problems for communication complexity of secure protocols. We show that, versus a private adversary controlling $\lfloor \frac{n}{2} \rfloor$ of the processors, $\Theta(n)$ computations of the n-ary addition modulo 2 function can be performed simultaneously using $O(n^2 \log n)$ bits of communication, while $\Omega(n^3)$ bits of communication are required if the computations are performed individually. We also show that, versus a somewhat stronger adversary (fail-stop mode) controlling $\lfloor \frac{n}{2} \rfloor$ of the processors, any k computations of the n-ary addition function over a field of size q costs $\Theta(kn^2 \log q)$; thus, for this fault-tolerance model for computing this problem, parallelization cannot help. This is in contrast with the tradeoff of Section 3, where paying with fault-tolerance does help reduce the communication.

In Section 5, we present a computation protocol that provides communication-efficient detectability. This protocol is compiled from a protocol for the privacy model. Specifically, our protocol performs the computation on exactly two sets of inputs simultaneously: the real secret inputs, and a collection of jointly-created check inputs. We suggest a technique by which the two computations proceed in *lock-step*, in such a way that one computation cannot be influenced without randomly affecting the other. At the end of the computation, the honest processors learn the (supposed) real output, the (supposed) check output, and the check inputs. We prove that if the honest processors accept the real output only when the check computation is correct, then the probability of undetected error can be made arbitrarily small. The bit complexity of this protocol is about the same as the cheapest known fault-tolerant private protocol although it protects against a much stronger adversary.

2 Model and Definitions

A group of n "processors" or "players" jointly evaluate an n-ary function over some finite field F with $|F|$ elements. The function is represented as an arithmetic circuit consisting of 2-ary addition gates, 1-ary scalar multiplication gates, and 2-ary multiplication gates. The class of all arithmetic circuits with exactly M multiplication gates is denoted C_M.

Each player is a probabilistic Turing Machine, and each pair of players is connected by a physically secure communication channel (the "non-cryptographic" or "unconditional" setting). Each

1 Introduction

Secure computation protocols allow the cooperative evaluation of an arithmetic circuit by a group of processors, where each processor initially knows one input to the circuit. The protocol must not leak information about other processors' secret inputs, even if some group of processors misbehaves. Misbehavior may be passive (e.g., pooling legally obtained information), or active (e.g., violating the protocol in an arbitrary coordinated attack). The basic problem of designing protocols for arbitrary arithmetic circuits was first solved for two processors by Yao [19], and for any number of processors by Goldreich, Micali, and Wigderson [11]. These solutions relied on unproven assumptions: the intractability of factoring, and the existence of trapdoor functions, respectively.

Starting with Ben-Or, Goldwasser, and Wigderson [6], and Chaum, Crépeau, and Damgård [7], more recent protocols have removed the reliance of multi-party protocols on unproven assumptions. These protocols rely instead on the physical assumption that untappable communication channels connect all pairs of processors; sometimes these are called "unconditional" or "non-cryptographic" protocols.

In this paper, we initiate the investigation of the communication complexity of unconditionally secure multi-party fault-tolerant protocols (where by fault-tolerant we mean able to withstand attack by a *constant fraction* of faulty processors[1]). Since communication is a crucial resource, it is both natural and important to understand its intrinsic limits by showing lower bounds, and to reduce it by presenting improved upper bounds. Previously, communication complexity for secure computation was studied only for the privacy model and only for the two-party case [12] [2] (see also [15] for results in the related model of cooperative two-party computation versus an eavesdropper). Recently, and independently, Kushilevitz [13] has studied the communication complexity of multi-party private addition (see our Lemma 2).

One basic question addressed by this paper is the "direct sum problem" for the communication complexity of secure computation (recently addressed in the "non-secure" setting by Feder, Kushilevitz, and Naor [10]), i.e., when can parallelization reduce the amortized complexity of secure computation? We show that the answer is "sometimes," "sometimes not," and "almost always," depending (of course) on security setting and fault-tolerance tradeoffs. Lower bound arguments allow us to prove that in one security setting (privacy only, assuming processors are curious but otherwise honest), parallelization can reduce amortized bit complexity, while in a somewhat stronger setting (unstoppability, assuming that processor may stop-fail and requiring the protocol to nevertheless continue) it cannot. If one is willing to pay a small price in fault-tolerance, however, we show a systematic technique for parallelizing many secure computation protocols at no increase in total bit complexity.

A related question addressed by this paper is how to protect against a very strong adversary without a big communication penalty. Existing protocols offer "resilience," which gives the honest processors the correct output despite arbitrary coordinated tampering by a subset of faulty processors (analogous to error correcting codes). When malicious behavior is infrequent, however, it suffices to notify the honest processors that tampering has occurred (analogous to error detecting codes). Once notified, the honest processors may repeat the computation (possibly switching to a resilient protocol) to obtain the correct output. We call this more modest protection "detectability,"

[1] For $O(\log n)$ faults only, a protocol due to Beaver, Feigenbaum, Kilian, and Rogaway [5] has bit complexity independent of circuit size (dependent on the input size and n); no known protocols tolerating a constant fraction of faulty players have this property.

Communication Complexity of Secure Computation
(Extended Abstract)

Matthew Franklin* Moti Yung†

Abstract

A secret-ballot vote for a single proposition is an example of a secure distributed computation. The goal is for n participants to jointly compute the output of some n-ary function (in this case, the sum of the votes), while protecting their individual inputs against some form of misbehavior.

In this paper, we initiate the investigation of the communication complexity of unconditionally secure multi-party computation, and its relation with various fault-tolerance models. We present upper and lower bounds on communication, as well as tradeoffs among resources.

First, we consider the "direct sum problem" for communication complexity of perfectly secure protocols: Can the communication complexity of securely computing a single function $f : F^n \rightarrow F$ at k sets of inputs be smaller if all are computed simultaneously than if each is computed individually? We show that the answer depends on the failure model. A factor of $O(\frac{n}{\log n})$ can be gained in the privacy model (where processors are curious but correct); specifically, when f is n-ary addition (mod 2), we show a lower bound of $\Omega(n^2)$ bits for computing f once and an upper bound of $O(n^2 \log n)$ for computing f $O(n)$ times simultaneously. No gain is possible in a slightly stronger fault model (fail-stop mode); specifically, when f is n-ary addition over $GF(q)$, we show an exact bound of $\Theta(kn^2 \log q)$ for computing f at k sets of inputs simultaneously (for any $k \geq 1$).

However, if one is willing to pay an additive cost in fault tolerance (from t to $t - k + 1$), then a variety of known non-cryptographic protocols (including "provably unparallelizable" protocols from above!) can be systematically compiled to compute one function at k sets of inputs *with no increase in communication complexity*. Our compilation technique is based on a new compression idea of polynomial-based multi-secret sharing.

Lastly, we show how to compile private protocols into error-detecting protocols at a big savings of a factor of $O(n^3)$ (up to a log factor) over the best known error-correcting protocols. This is a new notion of fault-tolerant protocols, and is especially useful when malicious behavior is infrequent, since error-detection implies error-correction in this case.

*Department of Computer Science, Columbia University, New York, NY 10027. Partially supported by an AT&T Bell Laboratories Ph.D. Scholarship

†IBM Research Division, T.J. Watson Center, Yorktown, NY 10598.